Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **472,** 3101–3114 (2017) doi:10.1093/mnras/stx2161
Advance Access publication 2017 August 23

Convolutional neural networks for transient candidate vetting in large-scale surveys

Fabian Gieseke,[1,2]★ Steven Bloemen,[3,4] Cas van den Bogaard,[1] Tom Heskes,[1] Jonas Kindler,[5] Richard A. Scalzo,[6,7,8] Valério A. R. M. Ribeiro,[3,9,10,11] Jan van Roestel,[3] Paul J. Groot,[3] Fang Yuan,[6,7] Anais Möller[6,7] and Brad E. Tucker[6,7]

[1]*Institute for Computing and Information Sciences, Radboud University, PO Box 9010, NL-6500 GL Nijmegen, the Netherlands*
[2]*Department of Computer Science, University of Copenhagen, Sigurdsgade 41, DK-2200 Copenhagen, Denmark*
[3]*Department of Astrophysics/IMAPP, Radboud University, PO Box 9010, NL-6500 GL Nijmegen, the Netherlands*
[4]*NOVA Optical InfraRed Instrumentation Group, Oude Hoogeveensedijk 4, NL-7991 PD Dwingeloo, the Netherlands*
[5]*Institute of Cognitive Science, University of Osnabrück, Wachsbleiche 27, D-49090 Osnabrück, Germany*
[6]*Research School of Astronomy and Astrophysics, Australian National University, Canberra, ACT 2611, Australia*
[7]*ARC Centre of Excellence for All-Sky Astrophysics (CAASTRO), University of Sydney, NSW 2006, Australia*
[8]*Centre for Translational Data Science, University of Sydney, Darlington, NSW 2008, Australia*
[9]*CIDMA, Departamento de Física, Universidade de Aveiro, Campus de Santiago, P-3810-193 Aveiro, Portugal*
[10]*Instituto de Telecomunicações, Campus de Santiago, P-3810-193 Aveiro, Portugal*
[11]*Department of Physics and Astronomy, Botswana International University of Science and Technology, Private Bay 16, Palapye, Botswana*

Accepted 2017 August 17. Received 2017 August 7; in original form 2016 December 5

ABSTRACT

Current synoptic sky surveys monitor large areas of the sky to find variable and transient astronomical sources. As the number of detections per night at a single telescope easily exceeds several thousand, current detection pipelines make intensive use of machine learning algorithms to classify the detected objects and to filter out the most interesting candidates. A number of upcoming surveys will produce up to three orders of magnitude more data, which renders high-precision classification systems essential to reduce the manual and, hence, expensive vetting by human experts. We present an approach based on convolutional neural networks to discriminate between true astrophysical sources and artefacts in reference-subtracted optical images. We show that relatively simple networks are already competitive with state-of-the-art systems and that their quality can further be improved via slightly deeper networks and additional pre-processing steps – eventually yielding models outperforming state-of-the-art systems. In particular, our best model correctly classifies about 97.3 per cent of all 'real' and 99.7 per cent of all 'bogus' instances on a test set containing 1942 'bogus' and 227 'real' instances in total. Furthermore, the networks considered in this work can also successfully classify these objects at hand without relying on difference images, which might pave the way for future detection pipelines not containing image subtraction steps at all.

Key words: methods: data analysis – techniques: image processing – surveys – supernovae: general.

1 INTRODUCTION

A number of large optical survey telescopes such as *Skymapper* (Keller et al. 2007), the *Palomar Transient Factory* (PTF; Rau et al. 2009) and Panoramic Survey Telescope and Rapid Response System (Pan-STARRS1) (Kaiser et al. 2010) are searching for transient events. New generation surveys will be able to scan large amounts of the sky faster and deeper allowing searches for extremely rare or hitherto undiscovered events, such as possible electromagnetic counterparts of gravitational wave sources (see e.g. Nissanke, Kasliwal & Georgieva 2013; Smartt et al. 2016). Those surveys will increase our statistical samples of more common events, such as supernovae, for experiments in cosmology and fundamental physics (Riess et al. 1998; Schmidt et al. 1998; Perlmutter et al. 1999).

The detection of rare transient events among the vast majority of relatively constant sources is an important yet challenging task. Most surveys use difference imaging to find variable stars and transients. This is usually achieved by performing a pixel-by-pixel subtraction of a pre-existing template image from the image of interest. Astrophysical sources that are variable or were absent in the template image remain, while constant sources – which represent

★ E-mail: fabian.gieseke@di.ku.dk

the vast majority of the detected sources – are removed at the pixel level. During the difference imaging process, the template image is aligned and resampled to take into account distortions in the target image, and a convolution is done to match the point-spread function in all regions of the image (see e.g. Alard & Lupton 1998; Alard 2000).

While this process, in principle, allows one to very efficiently find rare transient events, in practice many of the resulting images contain a large number of 'bogus' objects. These 'bogus' objects trigger source finding algorithms, but rather than being of astrophysical nature, they are, in reality, artefacts. Such artefacts can result from a variety of processes such as issues with image processing (e.g. bad alignment at the subtraction step, between the template and target images), detector imperfections, atmospheric dispersion and cosmic rays passing through the detector.

The number of potential detections can be very large, with thousands of events per night produced by current synoptic surveys, and millions of detections per night expected from future surveys such as the *Large Synoptic Sky Survey* (LSST; Ivezic et al. 2009). The classification of the detections as either 'real' or 'bogus' sources is a necessary but daunting task, which will become an even more serious problem in the future. Manual verification by humans is expensive and, most likely, impossible to conduct for the amounts of data expected. For this reason, automatic detection algorithms that yield both a high purity and a high completeness will play an essential role for future transient surveys.

Machine learning aims at constructing *models* that can perform classification tasks in an automatic manner (Hastie, Tibshirani & Friedman 2009; Murphy 2012). One particular subfield of machine learning techniques, called *deep learning* (LeCun, Bengio & Hinton 2015), has gained considerable attention during the past few years. Deep learning algorithms have successfully been applied to a variety of real-world tasks. Two recent trends have sparked the interest in such algorithms: (1) the dramatic increase of data volumes in almost any field, which, in turn, has produced a massive amount of labelled data that can be used to train and evaluate the models; and (2) the enormous increase in compute power, particularly due to massively-parallel devices such as graphics processing units (GPUs), which led to a significant reduction of the practical runtime needed to generate deep architectures (Coates et al. 2013).

This paper aims at improving the automatic identification of transient sources in astronomical images. A standard approach, usually implemented in current detection pipelines, is based on extracting features from photometric images, such as the fluxes of the detected sources. Given such a representation of the objects at hand, one resorts to well-established machine learning algorithms. The most widely used approaches are currently based on some kind of dimension reduction (e.g. by conducting a principal component analysis in the pre-processing phase or by extracting physically motivated features such as magnitudes or parameters that reflect the shape of the point spread function) and a subsequent application of classification methods such as *random forests*, *support vector machines*, *nearest neighbour techniques* or (standard) *artificial neural networks* (e.g. Bloom et al. 2012; Brink et al. 2013; Buisson et al. 2015; Goldstein et al. 2015; Wright et al. 2015; Morii et al. 2016). Some recent works also resort to deep neural networks (here, the term 'deep' refers to the number of hidden layers in a network, see Section 2 for details). For example, a recent approach resorts to a neural network with three hidden layers that is applied given physically-motivated features (Morii et al. 2016). Another approach is based on *recurrent neural networks* with up to two hidden layers, given time series data that stems from flux values extracted from different observations (Charnock & Moss 2016). Note that these schemes also resort to an explicit feature extraction step that is conducted in the pre-processing phase.

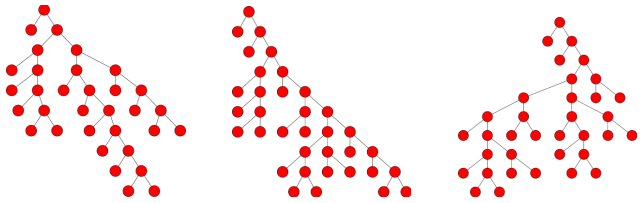


Figure 1. A random forest built for 50 training points. Each tree of the ensemble is built from top to bottom and at each node, slightly different 'splits' are used – resulting in different tree structures. The construction takes place until the leaves are *pure*, meaning that only patterns belonging to the same class are given in a single leaf (resulting in *less* than 50 leaves in this case). For splitting up the nodes, one resorts to different criteria such as the mean squared error for regression scenarios or the Gini index for classification tasks. A random forecast combines the predictions made by the individual trees.

This paper focuses on improving current detection pipelines and classification systems by means of *convolutional neural networks* (LeCun et al. 1989, 2015). In contrast to 'standard' deep architectures, convolutional neural networks do *not* rely on a feature extraction step conducted in the pre-processing phase. Instead, these models 'learn' good features based on the raw input image data. While convolutional neural networks have already been considered in the context of astronomy (Dieleman, Willett & Dambre 2015b; Kim & Brunner 2016), we present the first application of such models for the task of transient vetting. In this paper, we make use of a data set compiled in the framework of the *Skymapper* supernova searches (Scalzo et al. 2017).

2 BACKGROUND

In this section, we provide some machine learning background related to the techniques used in this work.

2.1 Random forests revisited

Random forests depict ensembles of individual classification trees (Breiman 2001; Hastie et al. 2009; Murphy 2012). In general, ensemble methods are among the most successful models in machine learning. This is especially true for random forests, which often yield high accuracies while being, at the same time, conceptually very simple and resilient against small changes of the involved parameter assignments. Since their introduction more than a decade ago, random forests have been extended and modified in various manners. A standard random forest consists of many individual trees (e.g. classification or regression trees), where each tree is built in a slightly 'different' way (see below) and the ensemble combines the benefits of all of them, see Fig. 1.

The trees of a random forest are usually constructed independently from each other. Each tree is built from top to bottom, where the root corresponds to all training instances and the leaves to subsets of the training data. During the construction, each internal node is recursively split into two children such that the resulting subsets exhibit a higher 'purity'. The overall process stops as soon as the leaves are 'pure' (i.e. they only contain patterns with the same label) or as soon as some other stopping criterion is fulfilled. The original way of building a random forest is based on subsets of

the training patterns, one subset for each tree to be built, called *bootstrap samples* (Breiman 2001). These subsets are drawn uniformly at random (with replacement) to obtain slightly different training sets and, hence, trees.

The quality of a node split is measured in terms of the gain in *purity*, which, in turn, is measured via different metrics depending on the desired outcome. Typical metrics include, for example, the *mean squared error* for the regression case or the *Gini index* for classification problems (Breiman 2001). For example, a pure split would be one yielding children containing only instances belonging to the same class, therefore, no further splits are required (Breiman 2001; Hastie et al. 2009; Murphy 2012).

Given a new, unseen instance, one can obtain a prediction for each single tree by traversing the tree from top to bottom based on the splitting information stored in the internal nodes until a leaf node is reached. The labels stored in this leaf are then combined to obtain the prediction for a single tree (e.g. by considering the mean for regression scenarios). The overall prediction of the random forest is based on a combination of the individual predictions. For regression scenarios, one usually simply averages the predictions. For classification settings, one can resort to a majority vote. In summary, the individual trees of a random forest can be seen as 'different' experts, whose opinions are combined to obtain a single overall prediction for a new instance.

2.2 Deep convolutional neural networks

Below we briefly introduce convolutional neural networks. For a more detailed description, we refer the reader to the excellent overview by LeCun et al. (2015) and the articles with applications to astronomical research (e.g. Dieleman et al. 2015b; Kim & Brunner 2016).

2.2.1 *Artificial neural networks*

Convolutional neural networks are special types of the standard *artificial neural networks* (ANNs; Hastie et al. 2009; Murphy 2012), which, in turn, consist of collections of interconnected nodes. In a nutshell, a multilayered artificial neural network is based on several *layers*, where the output of a given layer serves as input for the next layer. The first layer is called the *input layer* and the last layer the *output layer*. In between, there requires at least one *hidden layer*. For standard artificial neural networks, these layers are *fully connected*, meaning that all nodes of a given layer are connected to all nodes of the next layer (Fig. 2).[1]

The input layer is specified via the available input data. For example, given a feature vector $\boldsymbol{x} \in \mathbb{R}^d$ extracted from an image (e.g. a set of magnitudes), each of the feature values $x_1, \ldots, x_d$ corresponds to one of the input nodes of the input layer. Similarly, the output layer is determined by the learning task at hand. For a binary classification problem (e.g. 'bogus' versus 'real'), the output layer consists of two nodes that, for a given input instance, eventually output the class probabilities for each of the classes (note that in this special case, a single output node is sufficient since the classes are mutually exclusive).

The output of the first hidden layer is obtained via the weights $\mathbf{W}_1$ associated with the connections between the input layer and the first hidden layer. More specifically, the output of layer j is given by the transformation rule

$$\boldsymbol{x}_j = f(\mathbf{W}_j \boldsymbol{x}_{j-1} + \boldsymbol{b}_j), \tag{1}$$

where $\mathbf{W}_j$ is the weight matrix associated with the connections between layer $j-1$ and j, $\boldsymbol{b}_j$ a vector containing so-called *bias values* associated with layer j, and $f : \mathbb{R}^K \rightarrow \mathbb{R}^K$ a so-called *activation function* with K being the total number of neurons in layer j. Note that the transition from layer $j-1$ to j basically corresponds to applying a standard linear model followed by an element-wise application of the activation function f (Hastie et al. 2009; Murphy 2012). The dimensions of all involved vectors and matrices depend on the number of nodes and connections between the nodes. For example, for the transition from the input to the first hidden layer in Fig. 2, we have $\boldsymbol{x}_0 \in \mathbb{R}^4$, $\boldsymbol{x}_1 \in \mathbb{R}^6$, $\mathbf{W}_1 \in \mathbb{R}^{6\times4}$ and $\boldsymbol{b}_1 \in \mathbb{R}^6$. Popular choices for the activation function $f : \mathbb{R}^K \rightarrow \mathbb{R}^K$ are

(i) the *linear activation function* $[f(\boldsymbol{x})]_i = x_i$,
(ii) the *rectified activation function* $[f(\boldsymbol{x})]_i = \max(0, x_i)$,
(iii) or the *softmax activation function* $[f(\boldsymbol{x})]_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$

for $i = 1, \ldots, K$. Hence, the layers of a neural network iteratively transform the feature representation $\boldsymbol{x} \in \mathbb{R}^d$ of an input instance. Finally, the vector at the last hidden layer is transformed to only a single output neuron for standard regression scenarios or to multiple output neurons for classification or multivariate regression scenarios. Note that the different layers can resort to different activation functions, except for the last one, the output layer, which is somewhat restricted by the learning task at hand. For regression tasks, one usually makes use of a linear activation function, whereas the softmax activation function is a common choice for classification scenarios. Hence, given a new instance $\boldsymbol{x} \in \mathbb{R}^d$ for which one would like to obtain a prediction (e.g. if it is of type 'bogus' or 'real'), one consecutively applies the transformation rule (equation 1), which eventually yields the output vector $\boldsymbol{y}$. For regression scenarios, $\boldsymbol{y} \in \mathbb{R}^1$ corresponds to the prediction made by the network for the input $\boldsymbol{x}$, whereas the vector $\boldsymbol{y} \in \mathbb{R}^C$ contains the class probabilities given classification scenarios with C possible classes.

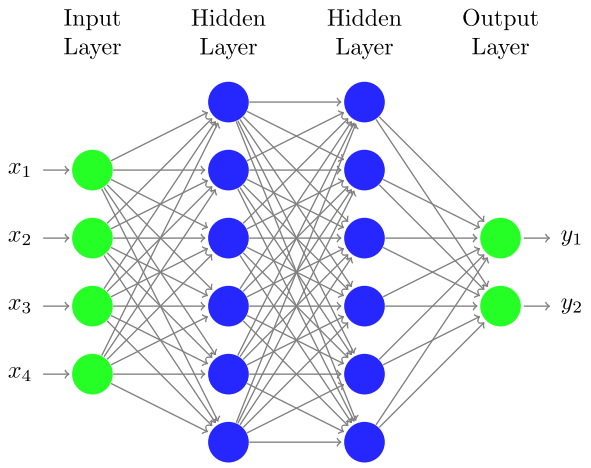


Figure 2. A standard fully connected artificial neural network with two hidden layers. The nodes of the network are called neurons and the output of each neuron corresponds to the weighted sum of its input neurons, transformed by an activation function. The output layer contains one neuron for each class and, given an input instance, outputs the corresponding class probabilities (in the case of classification scenarios).

[1] For convolutional neural networks, this is usually not the case except for the last layers (see below).

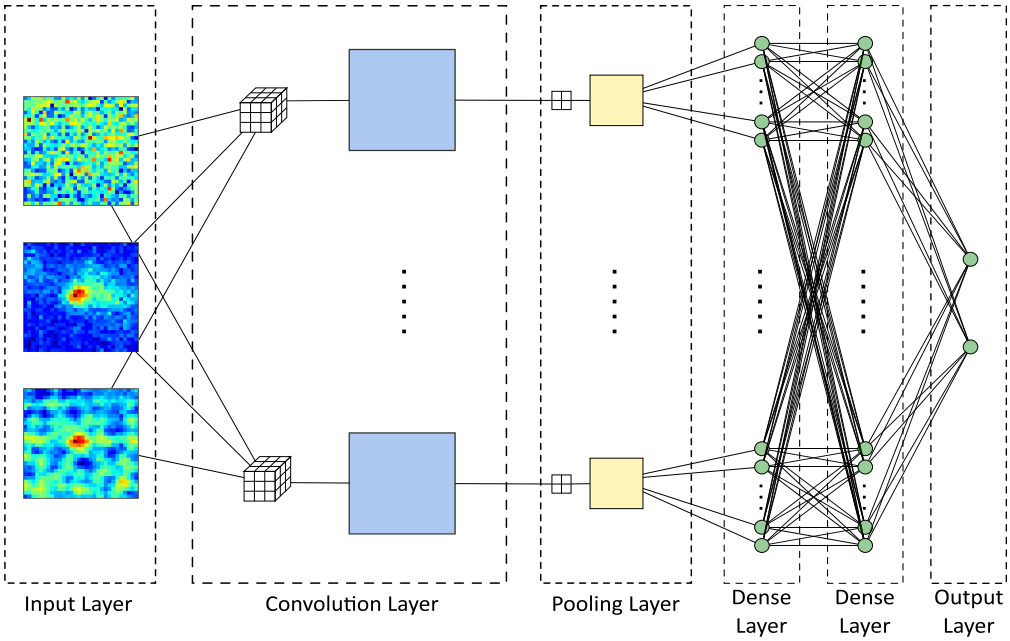


Figure 3. A convolutional neural network with one convolutional and one pooling layer, followed by two fully connected standard hidden layers and an output layer. Prior to the fully-connected hidden layers, the pixel-based feature maps are flattened, meaning that all pixel values of all feature maps of the previous layers are concatenated to form a single vector. Generally, by resorting to multiple convolutional and pooling layers, convolutional neural networks are capable of learning a hierarchical feature representation of the input instances – starting with simple features at early layers and more complex features towards the end.

Training such a neural network basically corresponds to finding weights such that the network performs well on new, unseen data (e.g. fewer misclassifications). Various optimization techniques can be applied (e.g. variants of gradient descent) so that the output of the network becomes more consistent with the class labels given for the training data. The so-called *learning rate* $\gamma > 0$ is a parameter used by many of the underlying optimization techniques that affects the size of the weight updates (e.g. similar to the step-size of standard gradient descent). The learning rate is a parameter that needs to be specified beforehand (as the network structure itself) and is usually tuned via grid search (i.e. various assignments are tested and one resorts to the training data to evaluate the induced quality).

Another important parameter is the number of *epochs*, which usually correspond to a full pass over the available training data or to processing a certain batch of a fixed size of training instances (most optimization techniques process the training instances iteratively). For the latter option, the *batch size* determines the number of instances processed per single epoch (Hastie et al. 2009).

2.2.2 *Convolutional neural networks*

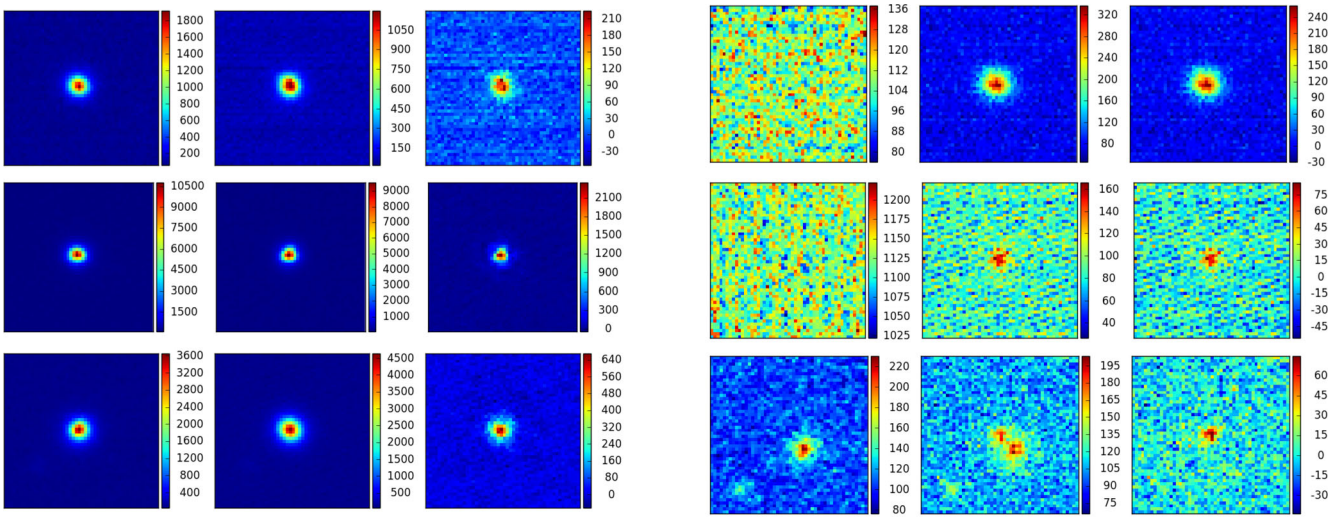
In recent years, so-called *deep networks* have become more and more popular. Here, the term 'deep' is related to the number of hidden layers. A special type of such deep architectures are convolutional neural networks, which consist of multiple layers of different types. Such networks have been successfully applied in the context of many application domains. We will focus on image-based input data for the description of convolutional neural networks.

A typical convolutional neural network with multiple hidden layers is shown in Fig. 3. As standard artificial networks, convolutional neural networks also exhibit an input and an output layer. Furthermore, the last hidden layers often correspond to standard fully connected dense layers as well. In contrast, the first hidden layers are conceptually very different and consist of various types of layers. The most prominent types, applied in this work, are *convolutional layers*, *pooling layers* and *dropout layers*:

(i) *Convolutional layers:* Such layers form the basis for convolutional neural networks and yield so-called *feature maps* by sliding a small window of weights across the input feature maps that stem from the previous layer (the input images of the input layer form the initial feature maps). In a nutshell, each feature map of a given layer stems from convolving all input feature maps using a set of *filters* (weight matrices), one filter for each input feature map. For example, in Fig. 3, we have, for each feature map in the first hidden layer, three filters of size 3×3 that are used to convolve all three input feature maps. The sum of all these convolved images correspond to the term $\mathbf{W}_j \boldsymbol{x}_{j-1}$ in equation (1). Afterwards, a matrix of bias values is added to this sum image, followed by an element-wise application of an activation function f.

(ii) *Pooling layers:* This is the second prominent type of layers. Pooling layers are used to decrease the number of learnable parameters (the filters/weight matrices). More precisely, such a pooling layer reduces the sizes of the feature maps by aggregating pixel values. For example, a max-pooling layer considers patches within each feature map and replaces each patch by the maximum value in that patch (e.g. in Fig. 3, each feature map of the previous layer is processed via patches of size 2×2, leading to new feature maps of half the size). Naturally, other operations can be applied such as taking the mean of a given pixel patch.

(iii) *Dropout layers:* These guard against *overfitting*, in which the trained network relies heavily on aspects of the training data that do not generalize well to new, unseen data. Dropout layers randomly omit hidden units by setting their value to zero with a user-defined probability $p \in (0, 1)$ such that other hidden units cannot fully rely on them (such techniques are also used in standard artificial neural networks). Thus, these layers force the network to rely on more units, preventing a reliance on noise or artefacts (dropout layers



(a) Bogus (b) Real

Figure 4. 'Best' case examples of three 'bogus' and three 'real'. The columns represent the template, target and difference images per field (from left to right) and the rows represent different fields. The different colours along with the colour bars illustrate the pixel intensities per image. For 'real' sources, there is usually no flux in the centre on the template image. However, there may be 'misleading' instances very close to the centre of the image (which, ideally, would no longer be present in the difference image). The majority of the instances in the data set are simple cases. As shown in our experiments, there is still a significant number of difficult instances outstanding, which can be very challenging for machine learning models to identify correctly. The unit of the colour scale is in counts.

are not shown in Fig. 3 since dropout basically only affects the underlying optimization process).

Therefore, convolutional layers aim to extract features that are somewhat invariant against translation. Note that there are also significantly less weights/connections for such convolutional layers (only the values given in the filters/weight matrices have to be learnt). Similarly, pooling layers make the network invariant against small transitions and reduce the number of nodes in the network (i.e. pixels in the feature maps). These two modifications often significantly increase the classification performance of such networks compared to standard fully connected artificial neural networks.

3 DEEP TRANSIENT DETECTION

In this section, we provide details of the different models considered for this paper in order to detect transients.

3.1 Imaging data

Our models were trained on early science operations, prior to 2015 April, imaging data from the *Skymapper Supernova and Transient Survey*. The difference imaging pipeline is described in more detail in (Scalzo et al. 2017). The main image processing tools used are the SWARP (Bertin et al. 2002) for astrometric registration and resampling to a common coordinate system, SExtractor (Bertin & Arnouts 1996) for source detection and photometry, and HOTPANTS (Becker 2015) for photometric registration and image subtraction. Data for each example include the *template* image, the *target* image and the *difference* image. In Fig. 4, we show examples of both 'real' and 'bogus' taken from the data set.

We trained our random forest models on features extracted from the images (Table 1), while the convolutional neural networks were applied directly on the images themselves. To evaluate the performance of each model, we split the available data into a training set and a test set of roughly equal size.

To make the most of our limited number of 'real' training instances, we enabled training on multiple distinct detections of the same transient candidate on different nights; this allowed us to sample more real variation in seeing and sky background level for the same candidate than we would if we included only one detection of each candidate. Because we expected the classifier results to also be affected by details of the host galaxy placement and morphology, which would be the same for multiple observations of the same transient, we placed all observations of the same object (e.g. a supernova) within the same partition (i.e. training or test); this choice enables our training methodology to make honest estimates of the generalization error to entirely new transient sources with different host galaxy properties. Apart from this constraint, the partition each individual transient candidate occupied was chosen at random. The training set contains 2237 instances (2010 'bogus' and 227 'real') of 851 distinct 'bogus' and 140 distinct 'real' sky objects. The test set contains 2236 instances (2,009 'bogus' and 227 'real') of 851 distinct 'bogus' and 141 distinct 'real' sky objects.

We removed those instances that were located less than 15 pixels from the edge of a CCD, in both the training and test sets, since the pixel cut-outs we use for our analysis (see Section 3.3) would be incomplete in these cases. This yields a final training set with 2162 instances (1939 'bogus' and 223 'real') and a test set containing 2169 instances (1942 'bogus' and 227 'real').

3.2 Baseline: Random forests and features

For the use of random forests, we extract various features from the imaging data (see Table 1). Most of the features are taken from Bloom et al. (2012). A large fraction of the features reflect properties of the source in the difference image, as well as some contextual information such as the presence of, and distance from, a nearby neighbour in the template image (Bloom et al. 2012). We supplemented the features above with new features intended to capture the global properties of the images – these flag obviously bad

Table 1. Features used as input for the random forest models.

Feature	Description
xsub	x coordinate on difference image, in pixels
ysub	y coordinate on difference image, in pixels
esub	ellipticity of source on difference image
thsub	direction of semimajor axis of source on difference image
fwhmsub	full width at half-maximum of all difference image sources
f4sub	flux within 4-pixel aperture in difference image
f8sub	flux within 8-pixel aperture in difference image
flagsub	SExtractor source flags in difference image
starsub	SExtractor star-galaxy score in difference image
xref	x coordinate on template image, in pixels
yref	y coordinate on template image, in pixels
eref	ellipticity of source on template image
thref	semimajor axis of source on template image
fwhmref	full width at half-maximum of all template image sources
f4ref	flux within 4-pixel aperture in template image
flagref	SExtractor source flags in template image
starref	SExtractor star-galaxy score in template image
enew	ellipticity of source on difference image
thnew	semimajor axis of source on difference image
fwhmnew	full width at half-maximum of all target image sources
f4new	flux within 4-pixel aperture in target image
flagnew	SExtractor source flags in target image
starnew	SExtractor star-galaxy score in target image
n2sig3	number of at least 2-sigma negative pixels in 3×3 box in difference image
n3sig3	number of at least 3-sigma negative pixels in 3×3 box in difference image
n2sig5	number of at least 2-sigma negative pixels in 5×5 box in difference image
n3sig5	number of at least 3-sigma negative pixels in 5×5 box in difference image
nmask	number of masked pixels in 5×5 box in the target image
Rfwhm	fwhmnew/fwhmref ratio
goodcn	surface density of detected sources on subtraction
subconv	direction of convolution (template-target or target-template)
nndref	distance in pixels to nearest neighbour source in template image
nndnew	distance in pixels to nearest neighbour source in target image
apsig4	signal-to-noise ratio of 4-pixel aperture flux in difference image
apsig8	signal-to-noise ratio of 4-pixel aperture flux in difference image
normrms	ratio of square root of isophotal area in difference image to fwhmsub
normfwhm	ratio of full width at half-maximumin difference image to fwhmsub
Rfref	signal-to-noise ratio of nearest counterpart in difference image
Raref	ratio of candidate semimajor axis to all sources in difference image
Reref	ratio of candidate ellipticity to all sources in difference image
Dthref	difference in candidate semimajor axis direction from all sources in difference image
Rfnew	signal-to-noise ratio of nearest counterpart in target image
Ranew	ratio of candidate semimajor axis to all sources in target image
Renew	ratio of candidate ellipticity to all sources in target image
Dthnew	target in candidate semimajor axis direction from all sources in target image

(e.g. trailed) images and subtractions. We also included SExtractor error codes and star–galaxy separation scores; the latter provide a redundant output from a different method (neural network) that may capture aspects of point sources not covered by our existing features.

As with many other machine learning models, the classification performance depends on the particular parameter assignments used to generate the random forest. However, random forests are usually very robust against small modifications, i.e. given reasonable parameter assignments, the validation performance is often very similar. The main parameters that need to be tuned are: (1) the number of estimators, (2) the number of features tested per split, (3) if bootstrap samples are used or not and (4) the stopping criterion used. Furthermore, variations of classical random forests exist such as the *extremely randomized trees* (Geurts, Ernst & Wehenkel 2006), which consider 'random' thresholds as potential splitting candidates.

For our analysis, we have tested different random forest variants and parameter assignments. However, for simplicity, we only report results of a single configuration (all others yielded very similar performances). In particular, we consider the Gini index to measure the impurity of the internal node splits, make use of 500 trees built using a bootstrap sample, resort to fully grown trees and test $\sqrt{d}$ features per node split, where d is the number of overall features considered.[2] We also tested various other parameter assignments, which all yielded very similar classification accuracies (as long as a sufficiently large amount of trees was considered).

3.3 Network structures and parameters

While convolutional neural networks have been successfully applied to several real-world tasks (see e.g. LeCun et al. 2015), choosing the best-performing network structure (w.r.t. the classification performance on the test set) is often based on trial and error. Very deep structures might be disadvantageous given 'simple tasks'. On the other hand, too simplistic structures might not be able to adapt to the learning task at hand and, thus, may yield unsatisfactory results as well. Therefore, the goal is to consider models that are complex enough to capture the characteristics of a learning task and that are, at the same time, not too complex. This is related to the so-called *bias-variance tradeoff* (Hastie et al. 2009), which describes the well-known problem in machine learning of finding models with the right complexity such that they perform well on unseen data, and to the optimization process itself. For example, deeper models generally exhibit more model parameters that need to be tuned/fitted, whereas in shallower networks one usually tunes less parameters. In this paper, we tackled this problem by starting with very simple and shallow convolutional networks and then increasing the complexity of the networks step-wise adding further convolutional layers. As will be shown in our experimental evaluation, simple convolutional neural networks already yield very promising classification results. Furthermore, due to their simplicity, one gains insight into how and why these networks perform so well. Additionally, the perfor-

[2] We use Python 2.7 and the `scikit-learn` package (version 0.18; Pedregosa et al. 2011) for processing and analysing the data. More precisely, we make use of the `sklearn.ensemble.RandomForestClassifier` class as the random forest implementation and initialize the model using the following parameters: `bootstrap=True`, `n_estimators=500`, `min_samples_split=2`, `criterion=''gini''` and `max_features=''sqrt''`.

Table 2. Network structures considered in this work ('fs' denotes the filter size of the convolutional layer, 'ps' the pooling size of the pooling layer and 'p' the dropout probability). A and B are parameters determining the input sizes of the layers.

Type	Size	Parameters
(a) Net1(A,B)		
input	$3 \times 30 \times 30$	
conv	$A \times 28 \times 28$	fs = (3,3)
maxpool	$A \times 14 \times 14$	ps = (2,2)
dropout	$A \times 14 \times 14$	p = 0.1
dense	B	
dropout	B	p = 0.5
dense	B	
dense	2	
(b) Net2		
input	$3 \times 30 \times 30$	
conv	$32 \times 28 \times 28$	fs = (3,3)
maxpool	$32 \times 14 \times 14$	ps = (2,2)
dropout	$32 \times 14 \times 14$	p = 0.1
conv	$128 \times 12 \times 12$	fs = (3,3)
maxpool	$128 \times 6 \times 6$	ps = (2,2)
dropout	$128 \times 6 \times 6$	p = 0.1
dense	512	
dropout	512	p = 0.5
dense	512	
dense	2	
(c) Net3		
input	$3 \times 30 \times 30$	
conv	$16 \times 28 \times 28$	fs = (3,3)
maxpool	$16 \times 14 \times 14$	ps = (2,2)
dropout	$16 \times 14 \times 14$	p = 0.1
conv	$32 \times 12 \times 12$	fs = (3,3)
maxpool	$32 \times 6 \times 6$	ps = (2,2)
dropout	$32 \times 6 \times 6$	p = 0.1
conv	$64 \times 4 \times 4$	fs = (3,3)
maxpool	$64 \times 2 \times 2$	ps = (2,2)
dropout	$64 \times 2 \times 2$	p = 0.1
dense	1000	
dropout	1000	p = 0.5
dense	1000	
dense	2	

mance may be improved by means of data augmentation and further pre-processing steps.

The network structures considered in this paper are presented in Table 2. Unless stated otherwise, the input layers correspond to the three input images that are available (i.e. template, target, difference). Each network contains at least one max-pooling, convolutional and dropout layers. The final layers are fully connected, followed by a softmax activation function to obtain class probabilities ('bogus' versus 'real'). The corresponding parameters for each layer are provided in the table. We considered 1000 training iterations for all networks without any data augmentation (see below) and 5000 for the ones with data augmentation. Here, a training iteration refers to processing a batch of 128 training images.

For the convolutional neural network approaches, we cropped the images to a size of 30×30 pixels and made use of the PYTHON package `nolearn` (version 0.6.1.dev0).[3] More precisely, we made use of the `nolearn.lasagne.NeuralNet` class and initialized the models with different parameters and layers (see Table 2). We complemented Net2 and Net3 with data augmentation steps that were conducted on the fly per training iteration (i.e. per batch of 128 training instances). In particular, we rotated each image by 90°, 95°, 100°, 180°, 185°, 190°, 270°, 275° and 280°. Subsequently, all 'real' instances were flipped horizontally and vertically. Note that we did not apply any translation augmentation steps since it is guaranteed that all 'real' instances are centred (up to 1 or 2 pixels).[4] Resampling of the augumented images was done in a manner to perserve the flux using the `scipy.ndimage.interpolation.rotate` function. Since the data augumentation step essentially yields a significantly larger data set, we increased the number of training iterations (5000 instead of 1000). For all networks, we resorted to the `nolearn` default settings to initialize the weights of all layers (i.e. Glorot with uniformly sampled weights; Glorot & Bengio 2010). Furthermore, to train the networks, we resorted to Adam updates with learning rate $\gamma = 0.0002$ (Kingma & Ba 2014). The overall process aimed to minimize the categorical cross entropy as objective with L2 regularization (`objective_lambda2=0.025`).

We made use of standard low-cost gaming GPUs, such as `Nvidia GeForce GTX 770`, to speed up the training and validating processes. Training the models was the most time-consuming phase of the two processes. Here, each training iteration (i.e. processing a batch of 128 images) took about 0.5 to 5 s depending on the network architecture and the particular GPU being used. Note, however, that the network models considered in this work can all be generated in a couple of minutes (e.g. 1000 training iterations for the shallow networks) or hours (e.g. 5000 training iterations and the deeper networks).

4 ANALYSIS

We compared the performance of the random forests approach, currently used in data processing pipelines, with those of different convolutional neural networks described above.

4.1 Experimental setup

All experiments resorted to data described in Section 3.1 to generate and evaluate the models. In the following, we assume that the label for a 'real' instance is +1 and the one for a 'bogus' instance is −1. We split the data into a 'training part', used for generating the corresponding models, and a 'testing part', used for the final evaluation of the models' classification performances. We also shuffle the training data set prior to training the different models. Note that none of the final test instances are shown to the model during the training phase. Therefore, the results indicate the performance of the classifiers on new, unseen data (given the same distribution of objects).

For fitting the random forest model, we resort to all instances given in the training set. For the convolutional networks, we use 95 per cent of the (potentially augmented) training set to actually fit the models, whereas five percent are used as a holdout validation set to monitor the objective values. It is worth mentioning that, for all networks considered, the objective values steadily decreased

[3] The `nolearn` package implements various wrappers for the `Lasagne` package, which, in turn, depicts a lightweight library for the well-known `Theano` package (see Nouri 2014; Dieleman et al. 2015a; Theano Development Team 2016, for details).

[4] We also conducted some experiments with very small translation steps, which, however, did not lead to an improvement w.r.t. the classification performance.

during the fitting process on both these parts of the training set up to a certain point, where additional epochs did not lead to any significant changes anymore. Furthermore, the objectives on both subsets were very close to each other, indicating that the convolutional neural networks did not overfit on the training set. Hence, both the employed regularization as well as the dropout layers seem to be effective measures against overfitting in this context.

Our data sets are imbalanced with regards to more 'bogus' than 'real' instances (approximately one 'real' instance out of 10 instances). The imbalance is relevant when assessing the performance of a given model. A classifier that simply assigns the class 'bogus' to all instances might already achieve very high accuracy, is, however, usually not helpful from a practical point of view since all 'real' instances would be missed. For this reason, it is crucial to consider an evaluation criteria that takes such class imbalanced into account. The evaluation criteria considered are based on different types of correct and incorrect classifications:

(i) *True positives (tp)*: Is defined as the number of 'real' instances that are correctly classified as 'real'.
(ii) *False positives (fp)*: Is defined as the number of 'bogus' instances that are misclassified as 'real'.
(iii) *True negatives (tn)*: Is defined as the number of 'bogus' instances that are correctly classified as 'bogus'.
(iv) *False negatives (fn)*: Is defined as the number of 'real' instances that are misclassified as 'bogus'.

The commonly used *accuracy* of a classifier is then given by $(tp + tn) \times (tp + tn + fp + fn)^{-1}$. Further, the *purity* (also called *precision*) is given by $tp \times (tp + fp)^{-1}$ and the *completeness* (also called *recall*) by $tp \times (tp + fn)^{-1}$. Another measure that combines the above numbers into a single number (still being meaningful for unbalanced data) is the so-called *Matthews correlation coefficient* (MCC; Matthews 1975):

$$\frac{tp \times tn - fp \times fn}{\sqrt{((tp + fp)(tp + fn)(tn + fp)(tn + fn))}}. \quad (2)$$

Here, an MCC of $+1.0$ corresponds to perfect predictions, -1.0 to total disagreement between predictions and the true classes, and 0.0 to a 'random guess'.

For a new instance, all models considered in this work output some kind of class probability value. More precisely, we consider softmax non-linearities for all convolutional neural networks and the mean predicted class probabilities of all trees for the random forest model (here, the class probability for a single tree is simply the fraction of the samples that belong to that class in a leaf). Unless stated otherwise, we will resort to the default 'threshold' of 0.5 to decide for a class label; we will analyse the influence of other thresholds at the end of our experimental evaluation.

From a practical point of view, training convolutional neural networks often takes much longer than training other machine learning models such as random forests. However, these models usually only need to be generated from time to time if new training data becomes available. Computing predictions for new, unseen instances takes considerably less time. This renders the networks described in this work applicable in the context of upcoming surveys with corresponding pipelines processing hundreds of thousands of candidate sources per night.

4.2 Results

We start by providing an overall comparison of various classification models and will subsequently analyse some of the models along with certain modifications in more detail.

4.2.1 *Classification performance*

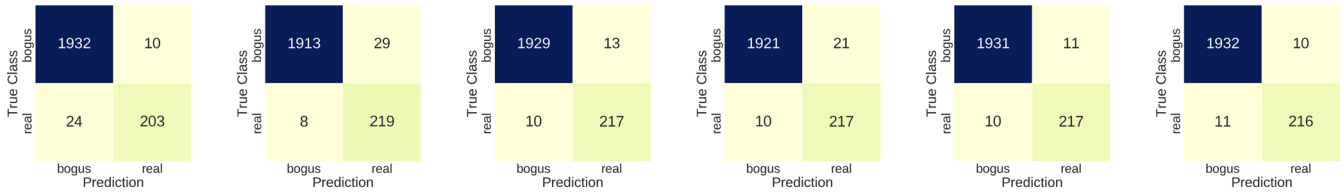
A meaningful evaluation measure (also for unbalanced data) are confusion matrices containing both the number of correctly classified instances as well as the number of misclassifications. The confusion matrices for various models, obtained via the test set, are shown in Fig. 5. It can be seen that each method performs reasonably well, but some of the models yield a significantly smaller number of misclassifications. In particular, one can make the following observations:

(i) *Random forest:* The standard random forest performs reasonably well, which indicates that the features extracted from the difference images capture the main characteristics of the learning task. In total, 34 instances are misclassified out of all 2169 instances, which corresponds to an MCC of 0.915. Note that a similar performance can be obtained by slightly different random forest models that stem from different splitting mechanisms or parameter assignments (see Section 3.2). However, none of these variants yielded an MCC better than 0.925.
(ii) *Shallow networks:* The conceptually very simple and shallow networks with only a single convolutional layer already yield a surprisingly good performance that is competitive with the one of the random forest. The MCCs for the Net1(32,64), Net1(64,128), and Net1(128,256) are 0.937, 0.951 and 0.933, respectively. We will investigate these networks and their classification performances in more detail below.
(iii) *Deeper networks:* The deeper networks with two or three convolutional layers yield a slightly better classification performance than the shallow networks. Applying data augmentation steps prior to training the networks seem to further reduce the number of misclassifications (see below). The MCC for Net2 and Net3 without any further data augmentation and transformation steps are 0.949 and 0.954, respectively. Net3 only misclassifies 21 out of the 2169 instances and, hence, 13 less than the random forest (in particular, it misses less 'real' sources that are assigned to the 'bogus' class).

It is worth mentioning that the classification performance is, in general, very similar for slightly different convolutional neural networks, i.e. neither the particular network structure nor the involved parameters seem to have a significant influence on the final classification performance. The conclusion one can draw at this point is that standard 'out-of-the-box' convolutional neural networks seem to be well suited for the task at hand and that even relatively simple networks yield a performance that is competitive with state-of-the-art approaches.

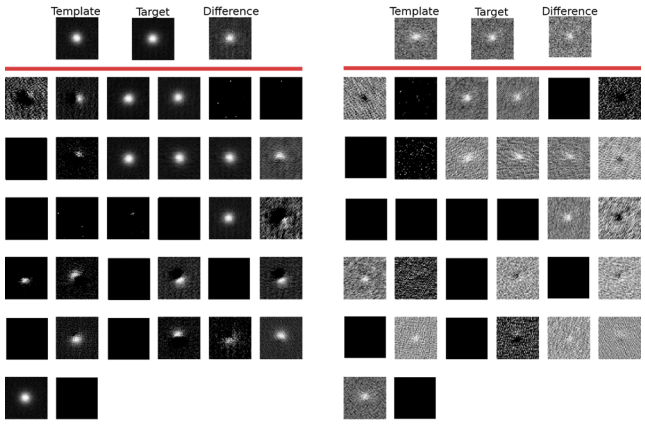
4.2.2 *Shallow networks*

Although being conceptually very simple, the shallow networks already yield a very good classification performance. This is actually surprising since the images are obtained under *different* observational conditions (such as the phases of the moon) that will lead to background levels that differ from image to image even though the same region/object is observed. Intuitively, 'subtracting' this background level in the pre-processing phase should simplify the learning problem and is, for this reason, also part of other approaches



(a) Random Forest (b) Net1(32,64) (c) Net1(64,128) (d) Net1(128,256) (e) Net2 (f) Net3

Figure 5. Confusion matrices for each of the considered models. While the random forest already achieves a good classification performance, the different networks, also the simple ones, seem to yield competitive or even slightly better results.



(a) "bogus" (b) "real"

Figure 6. Activations (feature maps) of the convolutional layer of Net1(32,64) given a 'bogus' and 'real' source, respectively. The top row shows the three input images that are available for each instance. Below the red line, the 32 feature maps are provided that are induced by the convolutional layer of Net1(32,64).

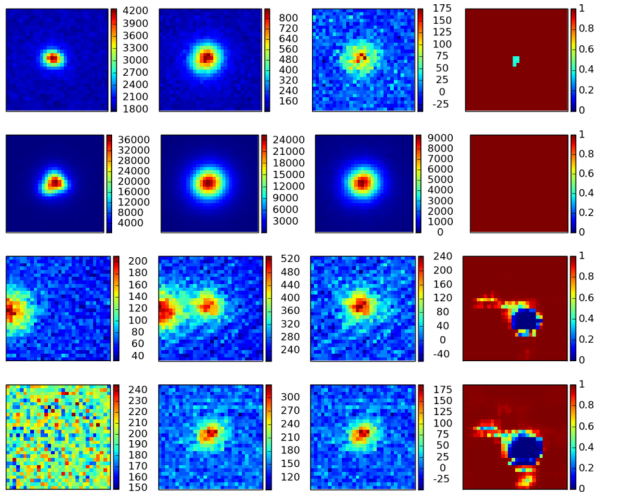


Figure 7. Input images along with occlusion maps (right column) for two 'bogus' (top) and two 'real' (bottom) sources. The different colours along with the colour bars illustrate the pixel intensities for each of the three left images per row, whereas they sketch the different probabilities for the occlusion map in the rightmost image per row. For the 'real' sources, the centres of the input images seem to be important, whereas for the 'bogus' instances, obscuring any part of the image appears to have little effect.

(Brink et al. 2013). However, we observed that such a normalization step in the pre-processing phase actually *decreases* the classification performances of the networks, both for the shallow and deeper models.

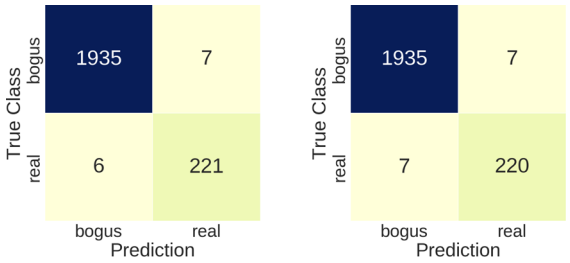
To investigate why the shallow networks already perform so well, we consider the simplest architecture, Net1(32,64), which only contains a single convolutional layer. In Fig. 6, the activations of the feature maps induced by a 'bogus' and a 'real' instance are shown. In both instances, there appears to be feature maps that activate on the background ('dark centre'), while other maps activate on different parts of the centre. This suggests that the network can distinguish between the source itself and the background, thus being able to classify images relatively unhindered by different levels of noise. We also consider *occlusion maps* (Zeiler & Fergus 2014) for Net1(32,64). To determine if a certain area of an image is important for the classification, one can mask this area and see how this affects the prediction of the model. In Fig. 7, occlusion maps are shown for several 'real' and 'bogus' instances. In these occlusion maps, the value of a pixel (x, y) represents the predicted probability of the correct class by the model, after all pixels in a 3×3 square centred on pixel (x, y) have been set to zero. Using the occlusion maps, one can gain insight into the way the model makes its predictions. For the 'real' sources, occluding pixels in the centre of an image causes misclassifications, while occluding the edges of the image has little to no effect. This is a good sign, as any other behaviour could indicate a reliance on artefacts or patterns in other regions of the image than the centre. For the 'bogus' instances, occluding any part of the image often makes little to no difference in classification. This indicates that the network learns that the absence of a source is an indicator for 'bogus' and as such, obscuring the 'bogus' source will still result in a correct classification.

In summary, as expected, the shallow networks seem to mainly focus on the centre (e.g. 'is there anything in the centre in the target image').

4.2.3 Data augmentation

One of the main challenges that needs to be addressed is the shift from the given training data to completely new, unseen objects. Since the number of 'real' sources is very small (e.g. only very few distinct supernovae objects are known and, hence, available for training), one has to develop a system that cannot only detect similar objects, but also new 'real' sources whose image representation is related, but different (e.g. a rotated version of an image containing a star, supernova or artefact). A simple yet effective approach to improve the generalization performance of convolutional neural networks is to augment the training data, see Section 3.3 for details and the particular augmentation steps conducted. Note that we only apply the augmentation on the training, the test set is not modified.

The results for the deeper networks, Net2 and Net3, enhanced with the data augmentation steps are shown in Fig. 8. Given that we increase the number of instances in the training data with augmentation, we also increase the number of training iterations from 1000



(a) Net2 (b) Net3

Figure 8. Confusion matrices for Net2 and Net3 with data augmentation steps conducted in the pre-processing phase.

to 5000. The confusion matrices in Fig. 8 show that the addition of a data augmentation step can further improve the classification performance with Net3 only causing 14 overall misclassifications for the test data set. We expect further data augmentations steps to be helpful as well in this context, see Section 5.

4.2.4 Analysis of misclassifications

Both Net2 and Net3 only misclassify a small number of test instances. In Figs 9 and 10, all misclassifications made by Net3 (with data augmentation) are shown. For the first type of error, 'bogus' objects misclassified as 'real', common examples are due to non-uniform background noise in the template and/or target images, or deficits in the template image that, after convolution, resemble point sources in the difference image. For the other type of error, 'real' instances misclassified as 'bogus', a single object (with multiple observations) is misclassified (top six images). This is very reasonable since it is generally very hard for any model to distinguish varying stars and multiple follow-up observations of a new supernova. However, from a practical perspective, such cases can easily be handled by flagging such a source as 'real' the first time it is observed (and correctly classified as 'real'). The remaining two (last two rows) depict a low signal-to-noise detection of a faint supernova near the core of its host galaxy, and an asteroid moving quickly enough to show a trail in the target image.

The two deeper convolutional neural networks yield significantly less misclassifications as the baseline random forest approach (see the Appendix for some misclassifications made by the random forest). Interestingly, the misclassifications differ slightly, i.e. the ones of Net3 do not form a subset of those misclassified by the random forest. We will see that this can actually be beneficial when combining the different classifiers.

4.2.5 Less input

The networks considered so far are trained on all three input images that are available for each instance. By providing all the data, the networks can automatically determine which input images are important (see discussion above concerning the weights). A natural question is whether a competitive performance can also be achieved using less input data. We consider two settings: (1) Using only the template and target images and (2) using only the difference image. Note that the latter setting usually forms the basis for other techniques that extract features from the difference images only.

We focus on the simplest network considered in this work, Net1(32,64), and the best-performing one, Net3 with data augmentation. The induced confusion matrices are shown in Fig. 11. By comparing this figure to Fig. 5, it can be seen that using only

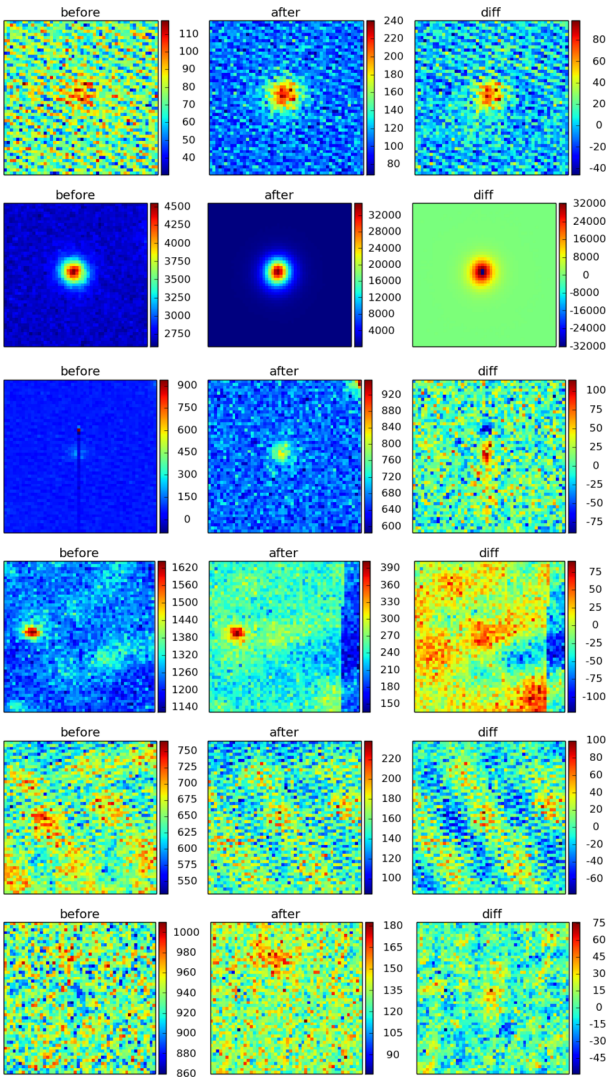


Figure 9. Misclassifications made by Net3 with data augmentation ('bogus' instances misclassified as 'real'). The different colours along with the colour bars illustrate the pixel intensities per image.

template and target images yields a competitive performance compared to using all three input images. This may seem surprising due to the majority of the existing schemes being based on difference imaging. The results, however, clearly indicate that the reduced set of input images is sufficient for approaching the task at hand. This depicts a desirable outcome since one might be able to omit image subtraction steps in future detection pipelines. Further, the networks trained using only the difference images yield a significantly worse classification performance. Hence, using only this type of information seems to be not enough for convolutional neural networks in this context.

4.2.6 Ensembles

A common way to improve the classification performance is to consider ensembles of different models. As mentioned above, random forests depict ensembles of classification or regression trees and usually yield a significantly better performance than the individual models. We consider two ensembles E1 and E2:

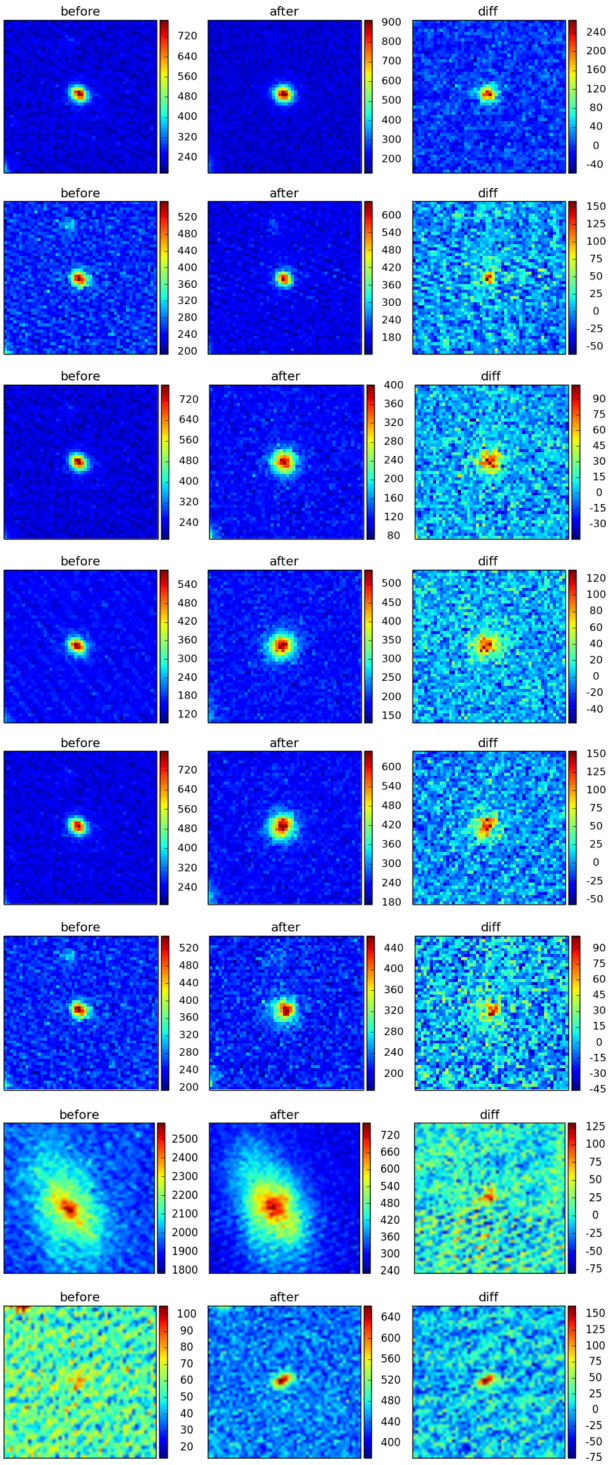


Figure 10. Misclassifications made by Net3 with data augmentation ('real' instances misclassified as 'bogus'). The different colours along with the colour bars illustrate the pixel intensities per image.

(i) *E1:* Net2 (data augmentation), Net3 (data augmentation), Net1(32,64) (template and target images only) and Net3 (data augmentation, template and target images only).

(ii) *E2:* Net2 (data augmentation), Net3 (data augmentation) and the random forest model.

The results are shown in Fig. 12. It can be seen that ensembling reduces the number of misclassifications. Furthermore, incorporating the random forest appears to be beneficial, potentially due to features that capture the characteristics of special cases.

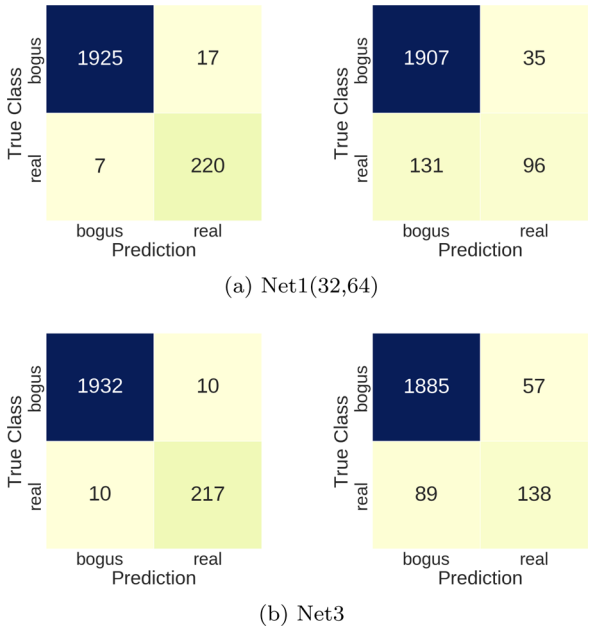


Figure 11. Classification performance of Net1(32,64) and Net3 (with data augmentation) in case only the template and target (left) or the difference images (right) are provided to the networks.

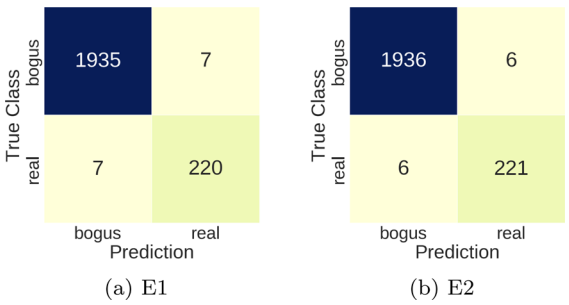


Figure 12. Confusion matrices for two ensemble classifiers. E1 combines three different neural networks. E2 two neural networks and a random forest.

The improvements over the best-performing single convolutional neural networks are really small and, due to the relatively small test data set, we do not argue that the ensembles outperform the individual classifiers. Nevertheless, the ensembles might exhibit a slightly better performance on completely new, unseen data since the combination of many different classifiers usually yield more 'stable' results.

4.2.7 Receiver Operating Characteristic (ROC) analysis

All results reported so far are based on the default threshold of 0.5 for deciding which class an instance should belong to given the probability scores. For random forests, this simply corresponds to a majority vote among the individual trees. For the convolutional neural networks, it means the class with the highest probability. In general, many more 'bogus' than 'real' instances are observed in practice and one might want to adapt the choice for the threshold. For example, one might prefer finding more 'real' sources at the cost of an increase in false positives (i.e. 'bogus' instances misclassified as 'real'). This naturally depends on the number of human experts

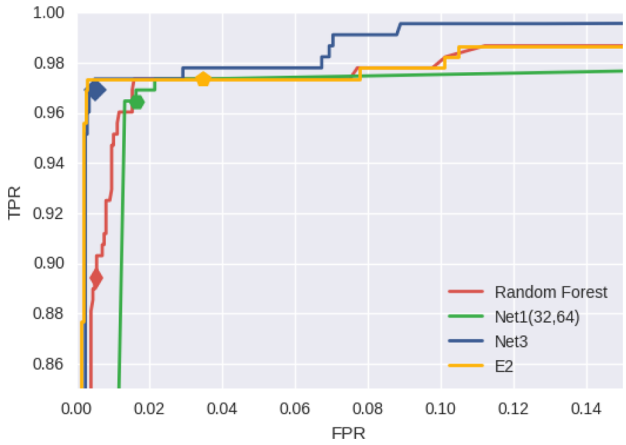


Figure 13. ROC curves for various models. The individual performances for a threshold of 0.5 are marked for each curve.

Table 3. AUC for the different models.

Model	AUC
Random forest	0.9907
Net1(32,64)	0.9914
Net3	0.9972
E2	0.9946

being available for manual inspection of all instances classified as 'real' sources by the model.

To quantify the performance of a model across a range of thresholds, one can make use of so-called *receiver operating characteristic* (ROC) curves (Fawcett 2006). Here, the recall $tp \cdot (tp + fn)^{-1} = tp \cdot P^{-1}$ is also called *true positive rate* (TPR), where P denotes the number of all positive ('real') instances. Accordingly, one can define the *false positive rate* (FPR) as $fp \cdot (fp + tn)^{-1} = fp \cdot N^{-1}$, where N corresponds to all negative ('bogus') objects. A classifier assigning only the class 'real' to all instances would therefore achieve an optimal TPR of 1.0, but also a potentially very large FPR. Ideally, one would like to have a large TPR and a small FPR; an ROC curve captures this trade-off.

In Fig. 13 the ROC curves for various models are shown. Of the models plotted, Net3 has the best performance, which we can verify by calculating the *area under the ROC curve* (AUC). The AUC values for the models are given in Table 3. To test the significance of the differences in AUC values, we apply two statistical tests for ROC curves: the so-called *DeLong* (DeLong, DeLong & Clarke-Pearson 1988) and the *bootstrap* (Hanley & McNeil 1983) methods. In both cases, we test the null hypothesis that the performance of both models is the same against the alternative hypothesis that Net3 performs better than the random forest (one-sided test). This results in p-values of 0.0359 and 0.0352, respectively, indicating that Net3 has a significantly better performance than the state-of-the-art random forest approach. Even though this improvement seems small, it could result in a large decrease in false positives due to the large number of transient candidates that are generated each night.

The confusion matrix for NET1(32,64) shows that we have a TPR of 0.956 for the standard threshold of 0.5. By moving right on the ROC curve, both the TPR and FPR increase and the decision of which FPR is still deemed acceptable is up to the user. In the case of transient vetting, the optimal threshold is determined by the capability to do follow-up studies on the possible transients and the willingness to search through a lot of extra 'bogus' candidates to find a couple more transients. Such decisions have to be made per project and based on the human resources that are available to manually check the output of the processing pipelines.

5 CONCLUSIONS AND OUTLOOK

We propose deep convolutional neural networks for the task of detecting astrophysical transients in future all-sky survey telescopes. The currently used state-of-the-art approach is based on feature extraction and a subsequent application of random forest algorithms. In our experimental evaluation, we demonstrate that even conceptually simple networks yield a competitive performance, which can be improved further via deeper architectures, data augmentation steps and ensembling techniques. It is also worth mentioning that the networks considered also perform well (or even better) by just using template and target images, i.e. the networks do not rely on image subtraction. This might pave the way for future classification pipelines not containing image subtraction pre-processing steps.

The machine learning models proposed in this work can be adapted and extended in various ways. Future telescope projects will produce significantly more data and we expect that taking such additional training instances into account will be beneficial to further improve the classification performances. The detection of extremely rare objects or artefacts will always depict a problem (even with better models due to many more objects being considered per night). Appropriate data pre-processing and augmentation steps conducted in the training phase might be one way to handle such instances correctly. In addition, adapting deep convolutional neural networks to the specific needs of the tasks at hand might be essential to cope with upcoming learning scenarios in this field (e.g. by considering specific loss functions that are suitable for extremely unbalanced data sets). We plan to investigate such important and interesting extensions in the near future.

ACKNOWLEDGEMENTS

FG and VARMR acknowledge financial support from the Radboud Excellence Initiative. VARMR further acknowledges financial support from Fundação para a Ciência e a Technologia (FCT) in the form of an exploratory project of reference IF/00498/2015, from Center for Research & Development in Mathematics and Applications (CIDMA) strategic project UID/MAT/04106/2013 and from Enabling Green E-science for the Square Kilometer Array Research Infrastructure (ENGAGE SKA), POCI-01-0145-FEDER-022217, funded by Programa Operacional Competitividade e Internacionalizaçêo (COMPETE 2020) and FCT, Portugal.

REFERENCES

Alard C., 2000, A&AS, 144, 363
Alard C., Lupton R. H., 1998, ApJ, 503, 325
Becker A., 2015, Astrophysics Source Code Library, record ascl: 1504.004
Bertin E., Arnouts S., 1996, A&AS, 117, 393
Bertin E., Mellier Y., Radovich M., Missonnier G., Didelon P., Morin B., 2002, in Bohlender D. A., Durand D., Handley T. H., eds, ASP Conf. Ser. Vol. 281, Astronomical Data Analysis Software and Systems XI. Astron. Soc. Pac., San Francisco, p. 228
Bloom J. S. et al., 2012, PASP, 124, 1175
Breiman L., 2001, Mach. Learn., 45, 5
Brink H., Richards J. W., Poznanski D., Bloom J. S., Rice J., Negahban S., Wainwright M., 2013, MNRAS, 435, 1047
Buisson du L., Sivanandam N., Bassett B. A., Smith M., 2015, MNRAS, 454, 2026

Charnock T., Moss A., 2016, ApJ, 837, L28
Coates A., Huval B., Wang T., Wu D. J., Catanzaro B. C., Ng A. Y., 2013, in Dasgupta S., McAllester D., eds, JMLR Proc. Ser. Vol. 28, 30th International Conference on Machine Learning (ICML). JMLR.org, p. 1337
DeLong E. R., DeLong D. M., Clarke-Pearson D. L., 1988, Biometrics, p. 837
Dieleman S. et al., 2015a, Lasagne: First release. Available at: http://dx.doi.org/10.5281/zenodo.27878
Dieleman S., Willett K. W., Dambre J., 2015b, MNRAS, 450, 1441
Fawcett T., 2006, Pattern Recognit. Lett., 27, 861
Geurts P., Ernst D., Wehenkel L., 2006, Mach. Learn., 63, 3
Glorot X., Bengio Y., 2010, in Teh Y. W., Titterington M., eds, Proc. Thirteenth International Conference on Artificial Intelligence and Statistics, Proc. of Machine Learning Research, p. 249
Goldstein D. A. et al., 2015, AJ, 150, 82
Hanley J. A., McNeil B. J., 1983, Radiology, 148, 839
Hastie T., Tibshirani R., Friedman J., 2009, The Elements of Statistical Learning. Springer-Verlag, Berlin
Ivezic Z. et al., 2009, Am. Astron. Soc. Meeting Abstr. 213, 460.03
Kaiser N. et al., 2010, in Stepp L. M., Gilmozzi R., Hall H. J., eds, Proc. SPIE Conf. Ser. Vol. 7733, Ground-based and Airborne Telescopes III. SPIE, Bellingham, p. 77330E
Keller S. C. et al., 2007, PASA, 24, 1
Kim E. J., Brunner R. J., 2016, MNRAS, 464, 4463
Kingma D. P., Ba J., 2014, CoRR, abs/1412.6980
LeCun Y., Boser B., Denker J. S., Henderson D., Howard R. E., Hubbard W., Jackel L. D., 1989, Neural Comput., 1, 541
LeCun Y., Bengio Y., Hinton G., 2015, Nature, 521, 436
Matthews B. W., 1975, Biochimica et Biophysica Acta, 405, 442
Morii M. et al., 2016, PASJ, 68, 104
Murphy K. P., 2012, Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge, MA
Nissanke S., Kasliwal M., Georgieva A., 2013, ApJ, 767, 124
Nouri D., 2014, nolearn: scikit-learn compatible neural network library. Available at: https://github.com/dnouri/nolearn
Pedregosa F. et al., 2011, J. Mach. Learn. Res., 12, 2825
Perlmutter S. et al., 1999, ApJ, 517, 565
Rau A. et al., 2009, PASP, 121, 1334
Riess A. G. et al., 1998, AJ, 116, 1009
Scalzo R. et al., 2017, PASA, 34, e030
Schmidt B. P. et al., 1998, ApJ, 507, 46
Smartt S. J. et al., 2016, MNRAS, 462, 4094
Theano Development Team 2016, preprint (arXiv:1605.02688)
Wright D. E. et al., 2015, MNRAS, 449, 451
Zeiler M. D., Fergus R., 2014, in Fleet D., Pajdla T., Schiele B., Tuytelaars T., eds, Proc. 13th European Conference on Computer Vision, ECCV 2014. Springer-Verlag, Berlin, p. 818

APPENDIX A: MISCLASSIFICATIONS

Figs A1 and A2 show misclassifications made by the random forest baseline. All false positive instances are given in Fig. A1, whereas only a subset is given for the false negatives in Fig. A2.

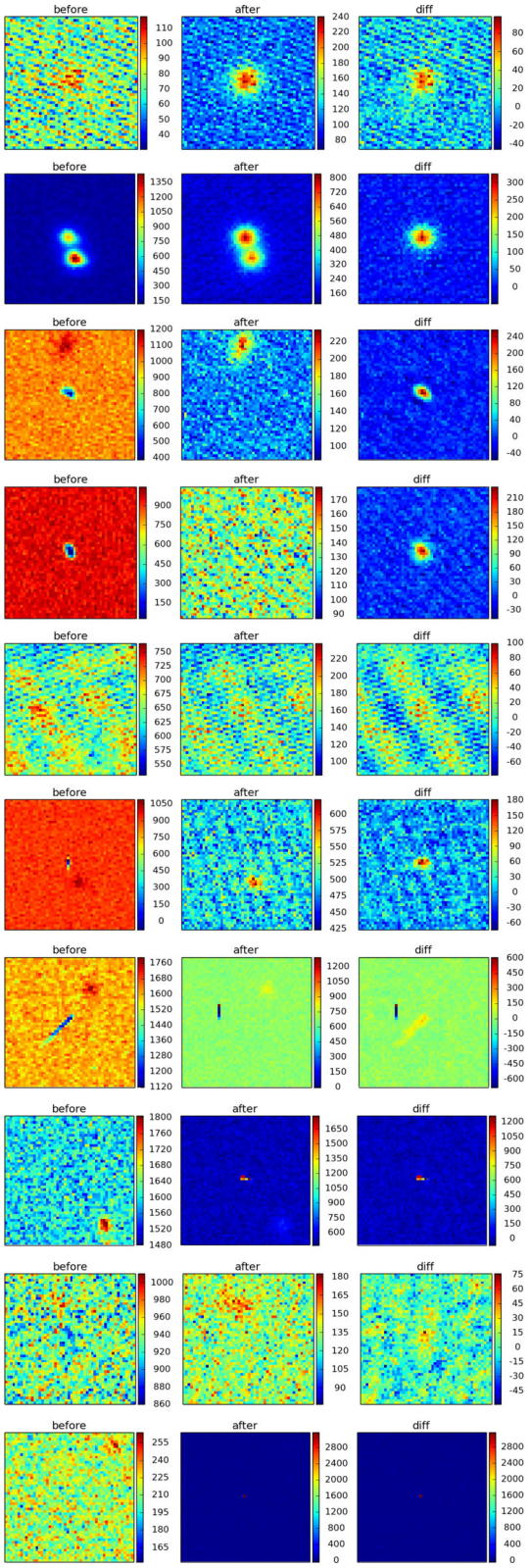


Figure A1. A subset of the 'bogus' objects misclassified as 'real' by the random forest model. The different colours along with the colour bars illustrate the pixel intensities per image.

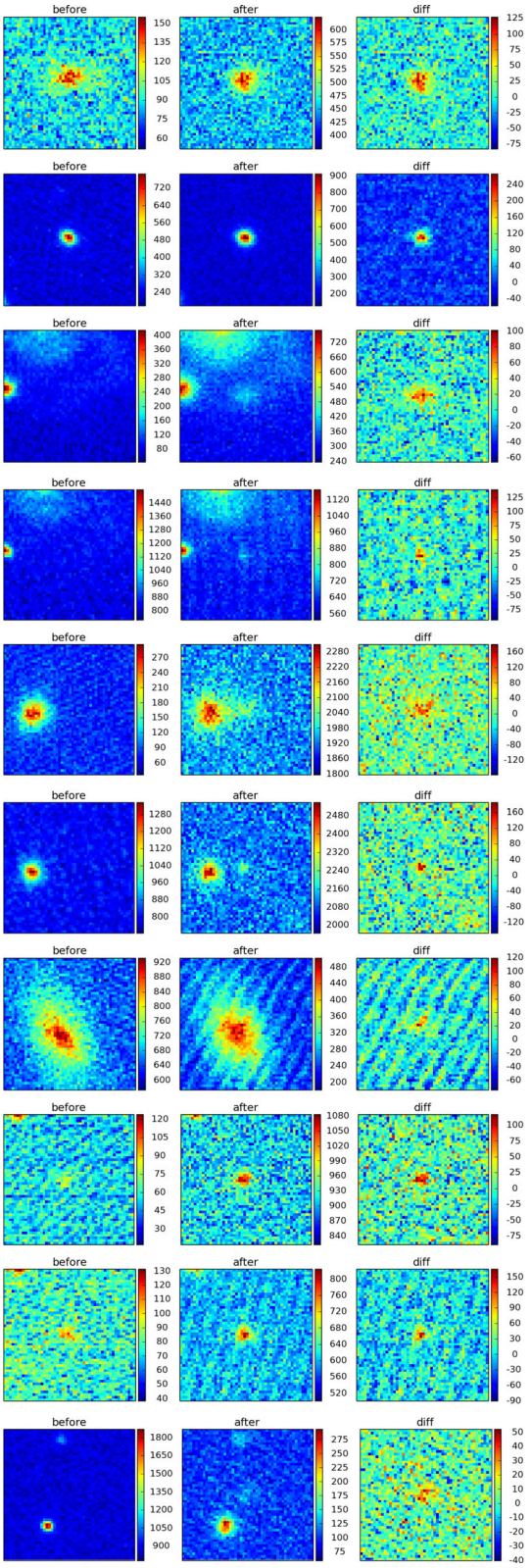


Figure A2. A subset of the 'real' objects misclassified as 'bogus' by the random forest model. The different colours along with the colour bars illustrate the pixel intensities per image.

This paper has been typeset from a TEX/LATEX file prepared by the author.